Incremental Maximum Flows for Fast Envelope Computation

Nicola Muscettola

NASA Ames Research Center
Moffett Field, CA 94035
mus@email.arc.nasa.gov

Abstract

Resource envelopes provide the tightest exact bounds on the resource consumption and production caused by all possible executions of a temporally flexible plan. We present a new class of algorithms that computes an envelope in *O(Maxflow(n, m, U))* where *n*, *m* and *U* measure the size of the flexible plan. This is an *O(n)* improvement on the best complexity bound for an envelope algorithm known so far and makes envelopes more amenable to practical use in scheduling. The reduction in complexity depends on the fact that when the algorithm computes the constant segment *i* of the envelope it makes full reuse of the maximum flow used to obtain segment *i-1*.

Resource Envelopes

The execution of plans greatly benefits from temporal flexibility. Fixed-time plans are brittle and may require extensive replanning due to execution uncertainty. Moreover, when plans must deal with uncontrollable exogenous events (Morris et al., 2001) temporal flexibility cannot be avoided. However, effective algorithms to build temporally flexible plans are rare, especially when activities produce or consume variable amounts of resource capacity. A major obstacle is the difficulty of assessing the resource needs across all possible plan executions.

Methods are available to compute resource consumption bounds (Laborie, 2001; Muscettola, 2002). In particular, (Muscettola, 2002) proposes a polynomial algorithm to compute a *resource envelope*, the tightest of these bounds. By being the tightest, resource envelopes can potentially save an exponential amount of search (through early backtracking and solution detection) when compared to using looser bounds. Also, methods that compute resource envelopes identify maximally matched sets of resource consumer/producers that balance each other for any plan execution. This and other structural information could be crucial in minimizing the search space and suggesting effective scheduling heuristics, potentially enabling new classes of highly efficient schedulers.

However, preliminary comparative studies of scheduling algorithms using envelopes appear not to show a computational advantage with respect to using more traditional heuristic methods based on fixed-time resource profiles (Pollicella et al., 2003). Since computing envelopes is more computational expensive than building a fixed-time profile, it is critical to ensure that the balance between computation cost and increased structural information extracted form the envelope is advantageous. Making the trade-off advantageous requires two complementary approaches. The first reduces the cost of computing an envelope; the second devises new envelope analysis methods to extract useful heuristics.

In this paper we address the problem of cost reduction. Currently, the resource envelope algorithm known to have the best asymptotic complexity (Muscettola, 2002) computes all piecewise-constant segments of the envelope through as many as $2n$ stages, where n is the number of events (start or end of activities) in the flexible plan. Each stage computes a maximum flow and therefore the overall complexity of the method is *O(n Maxflow(n,m,U))* where *m* is the number of temporal constraints between activities in the plan, *U* is the maximum level of resource production or consumption at some activity, and *Maxflow(n, m, U)* is the asymptotic cost of the maximum flow algorithm.

This staged method, however, can be significantly improved since at each stage a full maximum flow for the entire flexible plan is recomputed from scratch. Cost reduction could be obtained through an incremental flow method. Starting from the maximum flow at one stage, the maximum flow for the next stage is obtained by minimally reducing flow when deleting nodes and edges, and by minimally increasing flow when adding new nodes and edges (Kumar, 2003). However, without appropriately ordering flow reductions and increases, the asymptotic complexity may not improve (at it appears to be the case in (Kumar, 2003)).

In this paper we introduce an incremental method that provably computes an envelope in *O(Maxflow(n, m, U))* for a large class of maximum flow algorithms. This reduction of complexity is significant. Experimental analysis has shown that the practical cost of maximum flow is usually as low as $O(n^{1.5})$ (Ahuja et al., 1993). This compares well with *O(n log n)*, the cost of building resource profiles for fixed time schedules.

This paper is organized as follows. We first give a succinct introduction to the resource envelope problem and the staged envelope algorithm in (Muscettola, 2002). Next we present the new incremental algorithm and identify all sources of performance improvements. We then prove the complexity result, discuss implementation improvements when using preflow-push algorithms and conclude by discussing future work.

Staged Computation of Envelopes

In this section we outline the envelope problem and the staged algorithm that solves it. For a complete discussion, see (Muscettola, 2002).

Figure 1 shows an activity network with resource allocations. The network has two time variables per activity, a start event and an end event (e.g., e_{1s} and e_{1e} for activity A_1), a non-negative flexible activity duration link (e.g., *[2, 5]* for activity A_1), and flexible separation links between events (e.g., *[0, 4]* from e_{3e} to e_{4s}). Two additional events T_s and T_e define a time horizon within which all events occur.

Time origin, events and links constitute a Simple Temporal Network. To describe resource production and consumption each event also has an *allocation value* $r(e)$ (e.g., $r(e_{3s}) = -2$), a numeric weight that represents the amount of resource allocated when the event occurs. We will assume that all allocations refer to a single, multi-capacity resource. The extension to multiple resources is straightforward. If the allocation is negative an event e^- is a *consumer,* if it is positive e^+ is a *producer.* We assume that the temporal constraints are consistent which means that for any pair of events the shortest path $|e_1e_2|$ from e_1 to e_2 is well defined. Each event e can occur within its time bound, between the earliest time $et(e) = -|eT_s|$ and the latest time $lt(e) = |T_se|$. The triangular inequality $|e_1e_3| \leq |e_1e_2| + |e_2e_3|$ holds for any three events e_1, e_2 and e_3.

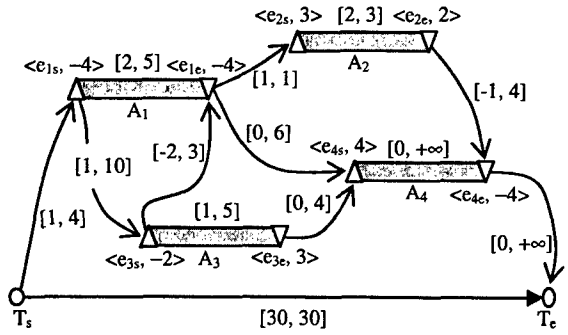

Figure 1: An activity network with resource allocations

Informally, a flexible plan is resource consistent if the duration and separation links induce appropriate necessary precedence relations between consumers and producers. These relations should guarantee that, when a consumer occurs, the total resource level due to consumers and producers that cannot occur after it must be at least as high as the new consumption. A similar condition applies to the correct occurrence of a producer. The full information on the necessary precedence relations is captured the *anti-precedence graph Aprec*, a graph that contains a path between any two events e_1 and e_2 if and only if $|e_1e_2| \leq 0$. Figure 2 depicts an anti-precedence graph of the network in Figure 1 with each event labeled with its time bound and resource allocation. We use anti-precedence graphs rather than the most customary precedence graphs (Laborie, 2001) to simplify the construction of the auxiliary maximum flow problem that, as we will see, is fundamental for the computation of envelopes.

We can now formally define a resource envelope. For any subset of events A, the *resource level increment* of A is $\Delta(A) = 0$ if $A = \varnothing$, and $\Delta(A) = \Sigma_{e \in A} r(e)$ if $A \neq \varnothing$. If S is the set of all possible consistent time instantiations for all events and t is a time within the time horizon, the resource level at time t for a specific time instantiation $s \in S$ is $L_s(t) = \Delta(E_s(t))$. Here $E_s(t)$ is the set of events e which occur at or before t in s. The *maximum resource envelope* is $L_{max}(t) = max_{s \in S} L_s(t)$ and the *minimum resource envelope* is $L_{min}(t)$

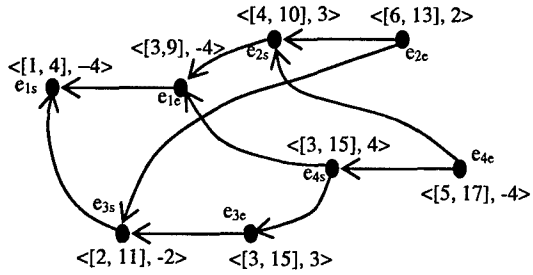

Figure 2: Anti-precedence graph with time bounds and resource allocations

$= min_{s \in S} L_s(t)$. Since L_{min} can be computed with obvious term substitution on the method that computes L_{max}, we only focus on L_{max}.

To compute the resource envelope at time t we partition all events into three sets depending on the position of their time bound relative to t: 1) the *closed events* C_t that must occur before or at t, i.e., such that that $lt(e) \leq t$; 2) the *pending events* R_t that can occur before, at or after t, i.e., such that $et(e) \leq t < lt(e)$; and 3) the *open events* O_t that must occur strictly after t, i.e., such that $et(e) > t$.

Any resource level increment $L_s(t)$ will always include the contribution of all events in C_t and none of those in O_t but may include only some subset of events in R_t, i.e., only those that are scheduled before t in s. It is possible to show that this subset must be a *predecessor set* $P \subseteq R_t$ such that if $e \in P$ and e' follows e in *Aprec*, then $e' \in P$. We call $P_{max}(R_t)$ the (possibly empty) predecessor set with maximum non-negative resource level increment.

The fundamental result reported in (Muscettola, 2002) is that $L_{max}(t)$ can be determined from the following equation.

Equation 1: $L_{max}(t) = \Delta(C_t) + \Delta(P_{max}(R_t))$

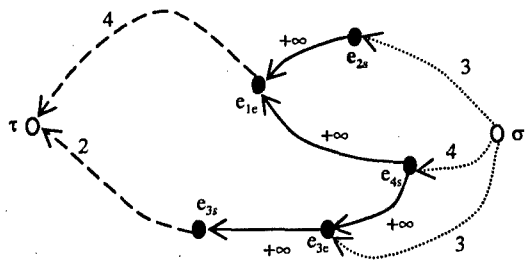

Figure 3: A resource increment flow network

Assuming that time bound information is available for all events, the computation cost of $\Delta(C_i)$ is $O(n)$. The cost of computing $P_{max}(R_i)$ determines the asymptotic cost of $L_{max}(t)$. We can compute $P_{max}(R_i)$ by solving a maximum flow problem on an auxiliary flow network $F(R_i)$, the *resource increment flow network* for R_i.

The formal definition of a resource increment flow network can be found in (Muscettola, 2002). As an example, Figure 2 gives $F(R_4)$ for the activity network in Figure 1. The network has a node for each event in R_4, an infinite capacity flow edge between two events for each edge in *Aprec* (see Figure 2), an edge from the source σ to a producer with capacity equal to the producer's allocation, and an edge from a consumer to the sink τ with capacity equal to the opposite of the consumer's allocation.

A complete discussion of maximum flow algorithms can be found in (Cormen, Leiserson and Rivest, 1990). Here we only highlight a few concepts that we will use in the following. A *flow* is a function $f(e_1, e_2)$ of pair of events in $F(R_i)$ that is skew-symmetric, i.e., $f(e_2, e_1) = -f(e_1, e_2)$, for each edge $e_1 \rightarrow e_2$ has a value no greater than the edge's capacity $c(e_1, e_2)$ (assuming capacity zero if the edge is not in $F(R_i)$), and is balanced, i.e., the sum of all flows entering an event must be zero. A *pre-flow* is a function defined similarly but that relaxes the balance constraint by allowing the sum of pre-flows entering a node to be positive. The total network flow is defined as $\Sigma_{e \in R_i} f(\sigma, e) = \Sigma_{e \in R_i} f(e, \tau)$. The maximum flow of a network is a flow function f_{max} such that the total network flow is maximum.

A fundamental concept in the theory of flows is the *residual network* for a particular flow, a graph with an edge for each pair of nodes in $F(R_i)$ with positive *residual capacity*, i.e., the difference $c(e_1, e_2) - f(e_1, e_2)$ between edge capacity and flow. Each residual network edge has capacity equal to the residual capacity. An *augmenting path* is a path connecting σ to τ in the residual network. The existence of an augmenting path indicates that additional flow can be pushed from σ to τ. Alternatively, the lack of an augmenting path indicates that a flow is maximum.

We can compute $P_{max}(R_i)$ according to the next theorem.

Theorem 2: [Theorem 1 in (Muscettola 2002)] *$P_{max}(R_i)$ is the (possibly empty) set of events that are reachable from the source σ in the residual network of some f_{max} of $F(R_i)$.*

From Equation 1 and Theorem 2 (Muscettola, 2002) derives a staged envelope algorithm as follows. Consider a time t_i corresponding either to the earliest or the latest time of some event. In a network with n events there are at most $2n$ such times. Since the envelope level can only change at one of these times, the algorithm computes a different level for each of them. At a particular t_i the algorithm determines the corresponding closed event set C_i and pending event set R_i, builds $F(R_i)$, computes one of its maximum flow using some appropriate maximum flow algorithm, determines $P_{max}(R_i)$ according to Theorem 2, and computes $L_{max}(t_i)$ according to Equation 1. It is easy to see that the worst-case time complexity of this algorithm is $O(n\ Maxflow(n, m, U))$ where $Maxflow(n, m, U)$ is the worst time complexity of the maximum flow algorithm used.

Incremental Computation of Envelopes

In the envelope algorithm previously described, maximum flows are recomputed from scratch for each $F(R_i)$. Assume that the times t_i are sorted in order of increasing value. To reduce the cost of computing the maximum flow for $F(R_i)$, we will follow the approach of reusing as much as possible of the maximum flow computed for $F(R_{i-1})$. Our theory is independent from specific maximum flow algorithms. Instead we build our argument on general properties of the resource increment flow networks.

As in (Muscettola, 2002) the fast envelope algorithm operates by the modular application of a maximum flow algorithm to a well defined sequence of resource increment flow network. The generality of the following theory opens the possibility of studying which flow algorithm could be most appropriate for different kinds of plan topologies.

Sources of Incremental Envelope Speedup

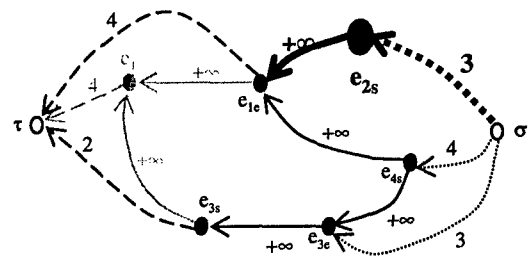

Figure 4: Incremental modification of a resource flow network

At time t_i the set of pending events can undergo two modifications. First, the events $\delta C_i = R_{i-1} - R_i$ move from R_{i-1} to C_i. These are events e such that $t_i = lt(e)$. Second, the events $\delta R_i = R_i - R_{i-1}$ move from O_{i-1} to R_i. These are the events e such that $t_i = et(e)$. For example, consider the activity network in Figure 1 and the process through which R_{i-1} for $t_{i-1} = 3$ is transformed into R_i for $t_i = 4$. This is described in Figure 4 where at time 4 the grayed part of the network is deleted and the emphasized part of the network is added. In particular, we have $\delta C_i = \{e_{1s}\}$ and $\delta R_i = \{e_{2s}\}$.

For completeness, we note that $F(\delta C_i)$ consists of node e_{1s} and edge $e_{1s} \rightarrow \tau$ while $F(\delta R_i)$ consists of node e_{2s} and edge $\sigma \rightarrow e_{2s}$. All other added and deleted edges are connectives from $F(R_{i-1} - \delta C_i)$ to $F(\delta C_i)$ (edges $e_{1e} \rightarrow e_{1s}$ and $e_{3s} \rightarrow e_{1s}$) and from $F(\delta R_i)$ and $F(R_{i-1} - \delta C_i)$ (edge $e_{2s} \rightarrow e_{1e}$).

The sets δC_i and δR_i satisfy the following fundamental properties.

Lemma 3: *δC_i is a predecessor set contained in R_{i-1}. δR_i is the complement of predecessor set R_{i-1} in R_i.*

Proof: We only give the proof for δC_i since the one for δR_i is analogous. We need to show that no predecessor of an event in δC_i can belong to its complement in R_{i-1}. In other words, given $e_1 \in \delta(C_i)$ and $e_2 \in R_{i-1} - \delta C_i$, using the definition of anti-precedence graph and predecessor set, it must be $|e_1 e_2| > 0$. From the definition of δC_i we have $lt(e_1) = t_i$ and $lt(e_2) \geq t_i + 1$[1]. From the triangular inequality applied to the latest times of e_1 and e_2, $lt(e_2) \leq lt(e_1) + |e_1 e_2|$, we deduce $|e_1 e_2| \geq lt(e_2) - lt(e_1) \geq t_i + 1 - t_i = 1 > 0$.□

Lemma 3 determines what flow edges are eliminated when δC_i is deleted and what are added when δR_i is added. In particular, we can only delete edges that enter events in δC_i or go from δC_i to τ. Similarly, we can only add edges that exit events in δR_i or go from σ to δR_i. Unlike previous proposals for incremental envelope calculation (Kumar, 2003), our method relies on events and edges exiting and entering the current flow network in a well defined order. This is the primary key to reducing complexity.

Directly related to Lemma 3 is the possibility of computing the maximum flow of $F(R_i)$ by incrementally modifying the flow of $F(R_{i-1})$, reusing both flow values and intermediate data structures across successive invocations of a maximum flow algorithm. We will prove that our flow modification operators guarantee the maximality of each intermediate flow. Maintaining intermediate flow maximality and reusing data structures are keys to reduce complexity for different kinds of maximum flow algorithms.

A final factor is minimizing the size of each intermediate flow network. We will show that as soon as the weight of an intermediate P_{max} is used in the envelope calculation, $F(P_{max})$ and all of its connecting edges can be safely eliminated from further consideration. This reduces flow network size and further contributes to cost reduction.

The argument to construct the fast envelope algorithm will proceed as follows. On the basis of Lemma 3, we first define flow transformation operators that apply to network additions/deletions occurring between time t_{i-1} and time t_i. The operators define the sequence of maximum-flow problems that need to be solved. Then we identify a flow separation property that, once applied to the sequence of maximum flow problems, further reduces the size of each step's maximum flow problem. Finally, we determine a recursive equation that computes $L_{max}(t_i)$ as a function of $L_{max}(t_{i-1})$ and the weights of event sets deduced from the application of flow transformation operators.

[1] For ease of exposition we assume discrete time although the theory applies also to continuous time with appropriate modifications.

Flow Modification Networks

The philosophy of each flow transformation operator is similar to that used by the flow augmentation method in maximum flow theory. However, we use this method more generally not only to augment flow but also to shift flow around the network and to reduce flow. The general idea is the following. Given a flow network F and one of its maximum flows f, an operator first defines an auxiliary flow transformation network F_T, then finds one of its maximum flows f_T, and finally produces a flow $f_{new} = f + f_T$. Each F_T consists of selected edges in the residual network of F for f. Since the properties of flows are preserved in the sum of a flow of F and a flow of its residual network, f_{new} is also a flow for network F.

Consider now the resource increment flow network $F(R_{i-1})$ at stage $i-1$ and assume that the set of new closed events δC_i is not empty. At stage i all events in δC_i and all of its incoming and outgoing edges will be deleted. This also means that any flow that at the end of stage $i-1$ enters δC_i will necessarily have to be zeroed, i.e., pushed back into $F(R_{i-1})$. The value of this flow is the sum of the residual capacities of all edges $e_1 \rightarrow e_2$ where $e_1 \in \delta C_i$ and $e_2 \in R_{i-1} - \delta C_i$. When pushed back, this flow can follow two routes. The first reaches τ through some non-saturated exiting edges of $F(R_{i-1} - \delta C_i)$. If after having followed the first route some flow is still flowing on some $e_1 \rightarrow e_2$ but the flow cannot reach τ any more, a second route allows reversing the remaining flow all the way to σ. We call this flow push-back operation a *flow contraction*. The first flow route corresponds to a *flow shift* and the second one to a *flow reduction*. For example, consider the network in Figure 4. Assume that at $t=3$ it is $f_{max}(e_{1s}, \tau) = 4$, $f_{max}(e_{1e}, \tau) = 1$ and $f_{max}(e_{3s}, \tau) = 2$. At $t=4$ the elimination of e_{1s} requires pushing back 4 units of flow. Three of these units can still reach τ by being shifted to $e_{1e} \rightarrow \tau$. Only one unit of flow needs to be pushed back to σ. If we pushed four units of flow back to σ without shifting (as in (Kumar, 2003)), later we would need to push again three units of flow from σ to τ to ensure flow maximality. This repetition of work affects worst-case asymptotic complexity.

Assume now that at stage i there is also a non-empty set δR_i of new pending events. Augmenting $F(R_{i-1} - \delta C_i)$ with the part of the resource increment flow network pertaining to δR_i yields $F(R_i)$. Assume now that $F(R_{i-1} - \delta C_i)$ is traversed by the flow resulting from flow contraction. Even if this flow is maximum for $F(R_{i-1} - \delta C_i)$, in general it will not be maximum for $F(R_i)$ since additional flow could be pushed through edges $\sigma \rightarrow e$ with $e \in \delta R_i$. We call this flow push-forward operation a *flow expansion*. If at every stage of flow contraction and flow expansion we guarantee flow maximality, we will obtain a maximum flow for $F(R_i)$ by moving a minimal amount of flow.

Flow Contraction

Let us call $f_{max,i-1}$ the maximum flow for $F(R_{i-1})$. In our discussion we ignore the structure of the flow sub-network for δC_i by using an auxiliary flow network $\underline{F}_{i-1}$ that redirects all flow entering δC_i into the sink τ. Formally, to obtain $\underline{F}_{i-1}$ we first delete from $F(R_i)$ all events in δC_i, together with all their incoming and outgoing flow edges. We then add an *auxiliary edge* $e_1 \rightarrow \tau$ for each set of component edges $e_1 \rightarrow e_2$ in $F(R_{i-1})$ such that $e_1 \in R_{i-1} - \delta C_i$ and $e_2 \in \delta C_i$. The capacity of the auxiliary edge $e_1 \rightarrow \tau$ is the sum of all $f_{max,i-1}(e_1, e_2)$. We call $\underline{f}_{max,i-1}$ a function over the edges of $\underline{F}_{i-1}$ where $\underline{f}_{max,i-1}(e_1,e_2)$ is equal to $f_{max,i-1}(e_1,e_2)$ if $e_1 \rightarrow e_2$ is not an auxiliary edge, and $\underline{f}_{max,i-1}(e_1,e_2)$ is equal to the edge's capacity if it is an auxiliary edge. It is easy to see that $\underline{f}_{max,i-1}$ is a maximum flow for $\underline{F}_{i-1}$. We call $\underline{Res}_{i-1}$ the residual network of $\underline{F}_{i-1}$ for $\underline{f}_{max,i-1}$.[1]

For example, consider the transformation at time $t_i=4$ of the network in Figure 4 and that at time $t_{i-1}=3$ it is $f_{max,i-1}(e_{1s}, \tau) = 4$, $f_{max,i-1}(e_{1e}, \tau) = 1$, $f_{max,i-1}(e_{3s}, \tau) = 2$, $f_{max,i-1}(e_{3s}, e_{is}) = 1$ and $f_{max,i-1}(e_{1e}, e_{1s}) = 3$. Then, $\underline{F}_{i-1}$ will not contain e_{1s} and all of its incoming and outgoing edged and will contain two additional flow edges of capacity $c(e_{1e}, \tau) = 3$ and $c(e_{3s}, \tau) = 1$. The maximum flow function $\underline{f}_{max,i-1}$ will have the same value as $f_{max,i-1}$ for all edges that already existed in $F(R_{i-1})$ and for the auxiliary edges it is $\underline{f}_{max,i-1}(e_{1e}, \tau) = c(e_{1e}, \tau) = 3$ and $\underline{f}_{max,i-1}(e_{3s}, \tau) = c(e_{3s}, \tau) = 1$. Note that we have only two edges in the residual network that are associated to auxiliary edges, $\tau \rightarrow e_{1e}$ with residual capacity 3 and $\tau \rightarrow e_{3s}$ with residual capacity 1, since both auxiliary edges are flow saturated.

We define a *flow shift network* $Shift_i$ as follows.

Flow shift network: ***$Shift_i$*** *is a flow network with the same nodes as* $\underline{Res}_{i-1}$. ***$Shift_i$*** *has a flow edge* $e_1 \rightarrow e_2$ *equal to a corresponding one in* $\underline{Res}_{i-1}$ *if* $e_1 \notin \{\sigma, \tau\}$ *and* $e_2 \neq \sigma$. *Finally, for each edge* $\tau \rightarrow e$ *in* $\underline{Res}_{i-1}$ *such that* $e \rightarrow \tau$ *is an auxiliary flow edge in* $\underline{F}_{i-1}$, ***$Shift_i$*** *has a corresponding edge* $\sigma \rightarrow e$ *of the same capacity.*

$Shift_i$ embodies the first route through which flow is pushed back. A maximum flow $f_{max,shift,i}$ for $Shift_i$ represents the maximum possible flow push-back. After this step, we produce a new flow $f' = \underline{f}_{max,i-1} + f_{max,shift,i}$ for $\underline{F}_{i-1}$. Now we need to formally define another auxiliary flow network, $Reduce_i$, to characterize the second flow push-back route, the one all the way back to σ. Let us call $Res(Shift_i)$ the residual network of $\underline{F}_{i-1}$ for $f' = \underline{f}_{max,i-1} + f_{max,shift,i}$. We define a *flow reduction network* $Reduce_i$ as follows.

Flow reduction network: ***$Reduce_i$*** *is a flow network with the same nodes as* ***$Res(Shift_i)$*** *and edges* $e_1 \rightarrow e_2$ *identical to* ***$Res(Shift_i)$*** *if* $e_2 \neq \tau$ *and either* $e_1 \notin \{\sigma, \tau\}$ *or* $e_1 = \tau$ *and the edge* $\tau \rightarrow e$ *corresponds to an auxiliary edge* $e \rightarrow \tau$ *in* $\underline{F}_{i-1}$.

[1] Note that if e_1 is a consumer, an edge $e_1 \rightarrow \tau$ will already exist before the introduction of a corresponding auxiliary edge. We tolerate the duplication of these edges since $\underline{F}_{i-1}$ is only intended as a formal device for the construction of $Shift_i$.

Using $Shift_i$ and $Reduce_i$, we define the **Flow_Contraction** operator needed by the incremental envelope algorithm.

Flow_Contraction$(F(R_{i-1}), f_{max,i-1}, \delta C_i, Aprec)$:
1) *Compute a maximum flow* $f_{max,shift,i}$ *for* ***$Shift_i$***;
2) *Compute a maximum flow* $f_{max,red,i}$ *for* ***$Reduce_i$***;
3) *Return* $f_{contr,i} = \underline{f}_{max,i-1} + f_{max,shift,i} + f_{max,red,i}$

We now prove that the operator keeps the flow maximum.

Lemma 4: *The flow* $f' = \underline{f}_{max,i-1} + f_{max,shift,i}$ *is maximum for* $\underline{F}_{i-1}$.
Proof: f' is a flow of $\underline{F}_{i-1}$. It is also maximum since by construction of $Shift_i$ it is $f_{max,shift,i}(\sigma, e) = 0$. Therefore $f'(\sigma, e) = \underline{f}_{max,i-1}(\sigma, e)$ and therefore f' is also maximum for $\underline{F}_{i-1}$.□

Lemma 5: $f_{contr,i}$ *is a flow for* $F(\mathbf{R}_{i-1} - \delta \mathbf{C}_i)$.
Proof: $f_{contr,i}$ is a flow for $\underline{F}_{i-1}$. For it to be a flow for $F(R_{i-1} - \delta C_i)$ it must be $f_{contr,i}(e, \tau) = 0$ if $e \rightarrow \tau$ is an auxiliary edge. If it were $f_{contr,i}(e, \tau) > 0$ for an auxiliary edge, by using the flow conservation constraint we could show that there must be a path from σ to τ, passing through $e \rightarrow \tau$, with all edges having positive flow. Therefore, there must be a flow-reducing path from τ to σ in the corresponding residual network. Such path is an augmenting path in the residual network of $Reduce_i$ for flow $f_{max,red,i}$, which contradicts the maximality of $f_{max,red,i}$.□

Theorem 6: $f_{contr,i}$ *is a maximum flow for* $F(\mathbf{R}_{i-1} - \delta \mathbf{C}_i)$.
Proof: This is clearly true if $f_{max,red,i}$ is a null flow since f' is maximum. If $f_{max,red,i}$ is not null, assume that $f_{contr,i}$ is not maximum. This yields an augmenting path Π from σ to τ in $F(R_{i-1} - \delta C_i)$ for $f_{contr,i}$. Since $\underline{f}_{max,i-i}$ is maximum, Π could only have appeared after the computation of $f_{max,shift,i}$. Since f' is maximum for $\underline{F}_{i-1}$, there must be at least one edge $e_1 \rightarrow e_2$ belonging to Π that does not belong to the residual network of $\underline{F}_{i-1}$ for f' otherwise $f_{max,shift,i}$ would not be maximum. Among these edges consider the one with either $e_2 = \tau$ or such that the suffix of Π going from e_2 to τ has positive residual capacity in $Shift_i$ for $f_{max,shift,i}$. A positive residual for $e_1 \rightarrow e_2$ implies that flow reduction pushed flow in the opposite direction, i.e., $f_{max,red,i}(e_2, e_1) > 0$. By back-tracing $f_{max,red,i}(e_2, e_1)$ we find a positive flow path for $f_{max,red,i}$ in $Reduce_i$ from σ to e_2. This can only happen if the capacity of the path in $Reduce_i$ is positive, which is equivalent to a prefix path with positive residual capacity in $Shift_i$ for $f_{max,shift,i}$. Tying the prefix and postfix at e_2 yields an augmenting path in $Shift_i$ for $f_{max,shift,i}$, impossible since $f_{max,shift,i}$ is maximum.□

Flow Expansion

To complete stage i we must now incorporate the event set δR_i to yield R_i and allow the computation of $P_{max,i} = P_{max}(R_i)$. Again, we define an incremental operation on an incremental residual flow network, the *flow expansion network*. The network is built on the residual network of $F(R_{i-1} - \delta C_i)$ for flow $f_{contr,i}$. We call this residual network $Res(Contr_i)$.

Flow expansion network: *$Expand_i$ is a flow network with nodes corresponding to those of R_i. $Expand_i$ contains all flow edges $e_1 \rightarrow e_2$ in $Res(Contr_i)$, all flow edges in $F(\delta R_i)$ and an infinite capacity edge $e_1 \rightarrow e_2$ for each anti-precedence edge between $e_1 \in \delta R_i$ and $e_2 \in R_{i-1} - \delta C_i$.*

Note that by construction $Expand_i$ is the residual network in $F(R_i)$ for $f_{contr,i}$. We now define the final operator needed by the incremental envelope algorithm, **Flow_Expansion.**

Flow_Expansion$(F(R_{i-1}-\delta C_i), f_{contr,i}, \delta R_i, Aprec)$:
1) Compute a maximum flow $f_{max,exp,i}$ for $Expand_i$;
2) Return $f_{max,i} = f_{contr,i} + f_{max,exp,i}$

Theorem 7: *$f_{max,i}$ is maximum for $F(R_i)$.*
Proof: $f_{max,i}$ is clearly a flow for $F(R_i)$. Moreover, $f_{max,exp,i}$ is maximum for $Expand_i$ and therefore there is no augmenting path in the corresponding residual network. The maximality of $f_{max,i}$ follows from the identity between the residual network of $Expand_i$ for $f_{max,exp,i}$ and the residual network of $F(R_i)$ for $f_{max,i}$.□

Flow Separation for P_{max}

We can achieve further performance improvements by minimizing the number of nodes and flow edges that need to be considered at each stage. During stage i, two P_{max} are computed: $P_{max,contr,i}$ after **Flow_Contraction**$_i$ and $P_{max,i}$ after **Flow_Expansion**$_i$. We know that each P_{max} is a predecessor set (i.e., it contains all of its successors in the anti-precedence graph), it is flow isolated (i.e., for each pair of events $e_1 \in P_{max}$ and $e_2 \in P^C_{max}$, $f_{max}(e_1, e_2) = 0$ and $f_{max}(e_2, e_1) = 0$) and has all exit edges saturated (i.e., $f_{max}(e, \tau) = c(e, \tau)$ for all $e \in P_{max}$) (Muscettola, 2002). This allows us to prove that $F(P_{max,i-1})$ can be ignored during the computation of **Flow_Contraction**$_i$ and $F(P_{max,contr,i})$ can be ignored during the computation of **Flow_Expansion**$_i$.

Let us consider each maximum flow operation executed at stage i. The first is flow shifting. Let us consider the properties of the set $\underline{P}_{max,i-1}$ of $\underline{F}_{i-1}$ that contains the events in $P_{max}(R_{i-1}) - \delta C_i$. $\underline{P}_{max,i-1}$ is a predecessor set since all events in δC_i are at the bottom of the anti-precedence graph for $F(R_{i-1})$. Moreover, due to added auxiliary edges in $\underline{F}_{i-1}$, the residual of the producers of $\underline{P}_{max,i-1}$ is the same as that in $P_{max,i-1}$ and therefore equal to $\Delta(P_{max}(R_{i-1}))$. $\underline{P}_{max,i-1}$ is still flow insulated and has all exit edges saturated. Assume that at some point during the flow shifting operation some additional flow reached an event $e' \in \underline{P}_{max,i-1}$. In order for at least part of such flow to reach τ there must be a postfix augmenting path that reaches τ from e'. But this is impossible since, being $\underline{P}_{max,i-1}$ a predecessor set, all postfix paths must remain inside $\underline{P}_{max,i-1}$ and all exit edges from $\underline{P}_{max,i-1}$ to τ are saturated. Therefore, any maximum flow algorithm that searches for augmenting paths can avoid doing so in $\underline{P}_{max,i-1}$. Moreover, in order for any excess flow pumped into events of $\underline{P}_{max,i}$ to achieve τ, that flow will have to be pushed back from $\underline{P}_{max,i-1}$ to $\underline{P}^C_{max,i-1}$. Therefore we can ignore $\underline{P}_{max,i-1}$ during flow shifting.

Since after flow shifting no flow has been changed for edges that touch $\underline{P}_{max,i-1}$, $\underline{P}_{max,i-1}$ maintains the flow insulation and saturation properties it had before flow shifting.

Considering now flow reduction, $f_{max,red,i}$ this can be computed by simply back-tracing flow in $\underline{F}_{i-1}$. Because of the flow insulation of $\underline{P}_{max,i-1}$, this back-tracing is either performed exclusively over edges connecting events in $\underline{P}^C_{max,i-1} = P^C_{max}(R_{i-1}) - \delta C_i$ or is confined within edges connecting events in $\underline{P}_{max,i,i}$. Since the entire flow that exits $\underline{P}_{max,i-1}$ will be back-traced, the entire $\underline{P}_{max,i-1} = P_{max}(R_{i-1}) - \delta C_i$ must belong to $P_{max,contr,i}$, the maximum predecessor set obtained after **Flow_Contraction**$_i$. Therefore the contribution of $\underline{P}_{max,i-1}$ to $P_{max,contr,i}$ can be known in advance without having to modify any flow of $F(\underline{P}_{max,i-1})$ during **Flow_Contraction**$_i$. Hence, $P_{max}(R_{i-1})$ can be taken out of consideration for future flow calculation as soon as it is computed.

Finally, we can use a similar argument to the one used for flow shifting to show that **Flow_Expansion**$_i$ can be performed entirely over $F(P^C_{max,contr,i})$, therefore allowing us to ignore $P_{max,,contr,i}$ at any future stage.

Incremental Computation of L_{max}

We are now ready to derive a recursive equations for the incremental calculation of $L_{max}(t)$ by transforming Equation 1 through the application of flow reduction and expansion.

Theorem 8: *$L_{max}(t)$ satisfies this recursive equation:*
if $t = t_1$
$L_{max}(t) = \Delta(C_1) + \Delta(P_{max}(R_1))$
if $t = t_i$ *and* $i > 1$
$L_{max}(t) = L_{max}(t_{i-1}) +$ *i*
$\Delta(\delta C_i \cap P^C_{max}(R_{i-1})) +$ *ii*
$\Delta(P_{max}(P^C_{max}(R_{i-1}) - \delta C_i)) +$ *iii*
$\Delta(P_{max}(\delta R_i \cup P^C_{max}(P^C_{max}(R_{i-1} - \delta C_i)))$; *iv*
if $t \neq t_i$, *then*
$L_{max}(t) = L_{max}(t\text{-}1)$.

Proof: $L_{max}(t)$ only changes when R_i changes, i.e., at a time t_i. Consider in turn the application of **Flow_Contraction**$_i$ and **Flow_Expansion**$_i$. Because of flow separation after **Flow_Contraction**$_{i-1}$, we have $P_{max,contr,i} = (P_{max}(R_{i-1}) - \delta C_i) \cup P_{max}(P^C_{max}(R_{i-1}) - \delta C_i)$. Analogously, after **Flow_Expansion**$_i$ we have $P_{max,i} = P_{max,contr,i} \cup P_{max}(P^C_{contr,i} \cup \delta R_i)$.

a) Flow_Contraction$_i$: the level after flow contraction, $L_{max,contr}(t_i)$ is the weight of the closed events after contraction and of $P_{max,contr,i}$. Since C_i and $P_{max,contr,i}$ are disjoint, $L_{max,contr}(t_i) = \Delta(C_{i-1} \cup \delta C_i \cup P_{max,contr,i}) = \Delta(C_{i-1}) + \Delta(\delta C_i \cup (P_{max}(R_{i-1}) - \delta C_i))) \cup \Delta(P_{max}(P^C_{max}(R_{i-1}) - \delta C_i))$. Since for any two sets A and B it is $A \cup (B - A) = B \cup (A - B)$, with B and $(A-B)$ being disjoint sets, we have $\delta C_i \cup (P_{max}(R_{i-1}) - \delta C_i) = P_{max}(R_{i-1}) \cup (\delta C_i - P_{max}(R_{i-1}))$. Hence, $\Delta(C_{i-1}) + \Delta(\delta C_i \cup (P_{max}(R_{i-1}) - \delta C_i)) = L_{max}(t_{i-1}) + \Delta(\delta C_i - P_{max}(R_{i-1}))$. Since $\delta C_i \subseteq R_{i-1} = P_{max}(R_{i-1}) \cup P^C_{max}(R_{i-1})$, it is easy to see that $\delta C_i - P_{max}(R_{i-1}) = \delta C_i \cap P^C_{max}(R_{i-1})$. This yields $L_{max,contr}(t_i) = L_{max}(t_{i-1}) + \Delta(\delta C_i \cap P^C_{max}(R_{i-1})) + \Delta(P_{max}(P^C_{max}(R_{i-1}) - \delta(C_i)))$, i.e., lines *i*, *ii* and *iii* in the theorem's statement.

b) **Flow_Expansion$_i$:** the only new increment comes from set $P_{max}(P^C_{max,contr,i} \cup \delta R_i) = P_{max}(P^C_{max}(R_{i-1} - \delta C_i) \cup \delta R_i)$ which yields line *iv* in the theorem's statement.□

```
Incremental_Resource_Envelope (N, Aprec(N))
{ 1: E := { Group events in the input set N into entries Et with three
          members: a time t and two lists earliest and latest. Event
          e∈N is included in Et.earliest if et(e) = t and in Et.latest if
          lt(e) = t. Sort the Et in increasing order of t. }
  2: Lmax := {<-∞, 0>}  /* Maximum resource envelope. */
  3: tcur := 0;         /* Current time */
  4: Lold := 0;         /* Envelope level at previous iteration. */
  5: Lnew := 0;         /* Envelope level at current iteration. */
  6: Pmax := ∅;         /* Maximum increment predecessors. */
  7: Fcur := ∅;         /*Resource increment flow graph with associated
                        maximum flow */
  8: Ecur := ∅;         /* Entry from E at tcur. */
  9: while (E is not empty)
 10:  {Ecur := pop(E);
 11:  tcur := Ecur.t;
 12:  Lnew := Lold + weight (intersection (Events(Fcur), Ecur.latest));
 13:  Fcur := Flow_Contraction (Fcur, Ecur.latest, Aprec(N));
 14:  <Pmax, Fcur> := Extract_P_Max (Fcur);
 15:  Lnew := Lnew + weight (Pmax);
 16:  Fcur := Flow_Expansion (Fcur, Ecur.earliest, Aprec(N));
 17:  <Pmax, Fcur> := Extract_P_max (Fcur);
 18:  Lnew := Lnew + weight (Pmax);
 19:  Lmax := append (Lmax, <tcur, Lnew>);
 20:  Lold := Lnew;
      }
  return Lmax;
}
```

Figure 5: **Incremental envelope algorithm**

Figure 5 shows the pseudocode of the algorithm. The functions **Flow_Contraction** and **Flow_Expansion** receive as arguments the current flow network $\mathbf{F_{cur}}$, which includes the current maximum flow, the incremental set of events that need to be added/deleted **Ecur.{earliest,latest}**, and the anti-precedence graph **Aprec(N)** for the set of all events **N** in the plan. **Aprec** carries the topological information needed to expand the flow network.

Given the current flow network and its maximum flow both stored in $\mathbf{F_{cur}}$, **Extract_P_max** returns both its maximum increment predecessor set $\mathbf{P_{max}}$ and the restricted network and flow resulting from the elimination of the $\mathbf{P_{max}}$. Comparing with the formula for $L_{max}(t_i)$ described by Theorem 8, line 12 in the algorithm computes *i+ii*, line 15 adds *iii* and line 18 adds *iv*. Note that the pseudocode represents a methodology, i.e., a class of algorithms. Specific algorithms can be implemented selecting different maximum flow algorithms in **Flow_Contraction** and **Flow_Expansion**. As we shall see the worst-case time complexity of a specific instantiation of the methodology is the same as that the maximum flow algorithm used.

Complexity Analysis

The following complexity analysis applies to a large number of maximum flow algorithms used for **Flow_Contraction** and **Flow_Expansion**. Each algorithm has a *complexity key*, i.e., a measurable entity whose static properties or dynamic behavior during the algorithm's computation determines the algorithm's time complexity. Table 1 (adapted from (Ahuja, Magnanti and Orlin, 1992)) reports the time complexity and complexity key of several maximum flow algorithms.

Algorithm	Time Complexity	Complexity Key
Labeling	$O(nmU)$	Total pushable flow
Capacity scaling	$O(nm \log U)$	Total pushable flow
Successive shortest paths	$O(n^2m)$	Shortest distance to τ
Generic Preflow-push	$O(n^2m)$	Distance label
FIFO reflow-push	$O(n^3)$	Distance label

Table 1: Complexity of maximum flow algorithms

The *Labeling* and *Capacity Scaling* algorithms are based on the original Ford-Fulkerson method. Their complexity depends on the strict monotonicity of the flow pushed during each algorithm's iteration and on the fact that the *total pushable flow* is bound by nU where U is the maximum capacity of an edge $\sigma \rightarrow e$ or $e \rightarrow \tau$. The *successive shortest paths* class of algorithms is based on the original Edmonds-Karp algorithm. The complexity depends on the fact that flow is pushed through augmenting paths of monotonically increasing length. The complexity key for this class of algorithms is the *shortest distance to* τ for each event e. For these algorithms it is crucial to demonstrate that the distance function $d(e)$ increases by at least one unit after each iteration.

Finally, preflow-push algorithms such as *generic preflow-push* and *FIFO preflow-push* (Goldberg and Tarjan, 1988) maintain a *distance labeling* $\underline{d}(e)$. These algorithms use purely local operations that push excess flow available at node e_1 through active edges $e_1 \rightarrow e_2$ such that $\underline{d}(e_1) = \underline{d}(e_2) + 1$. When excess flow exists at some node and no edge is active, the node's distance labeling is increased by the minimum amount that activates some edge. This allows more flow to be pushed. The complexity of preflow-push depends on creating a valid labeling at each iteration and on the fact that for each node the distance labeling is monotonically increasing up to $2n-1$.

To derive the complexity of our incremental envelope construction methodology, we analyze how each of the

complexity keys described before are modified across each contraction and expansion stage. Our goal is to show that when moving across stages the invariant properties of the complexity keys vary consistently to what is required by the maximum flow algorithms within each phase.

Consider first the cumulative cost of computing all flows over $2n$ stages respectively for $f_{max,shift,i}$, $f_{max,red,i}$ and $f_{max,exp,i}$. First note that at each stage $f_{max,red,i}$ can be computed by flow back-tracing through a backwards depth first search on the resource increment flow network. Since this can cost up to $O(m)$, the total cost of computing flow reduction is $O(nm)$ and is therefore smaller than the cost of applying a regular maximum flow algorithm. Therefore we focus on the cost for the cumulative $f_{max,shift,i}$ and $f_{max,exp,i}$, respectively $F_{shift} = \Sigma_i f_{max,shift,i}$ and $F_{exp} = \Sigma_i f_{max,exp,i}$.

Lemma 9: *Neither F_{shift} nor F_{exp} are greater than $n\,U$.*
Proof: We develop the argument for F_{shift} since the one for F_{exp} is similar. Consider the total capacity of the edges $\sigma \rightarrow e$ entering the auxiliary flow network $Shift_i$. Its upper bound is the total capacity of edges $e^- \rightarrow \tau$ with $e^- \in \delta(C_i)$. After iteration i all nodes in δC_i are eliminated from further consideration, hence flow can go through each $\sigma \rightarrow e$ in $Shift_i$ only during iteration i. Therefore, the total flow is upper bounded by $\Sigma_i |\delta C_i|\, U = n\, U$.□

Note that the argument above does not hold for F_{exp} if we do not use flow shifting but the flow is simply reduced and then expanded again (Kumar, 2003). In this case the same flow could be pushed up to n times and the cost of F_{exp} would be $O(n^2U)$. This would not improve on the staged envelope algorithm in (Muscettola, 2002).

Now, let us focus on the second complexity key, the shortest distance function $d(e)$ from e to τ. We will focus on how $d(e)$ changes at the beginning and the end of successive $f_{max,shift,i}$ and $f_{max,exp,i}$ computations. Let us call $d^0_{shift,i}(e)$ and $d^f_{shift,i}(e)$ the distances at the beginning and at the end of flow shifting for iteration i. We define similarly $d^0_{exp,i}(e)$ and $d^f_{exp,i}(e)$.

Lemma 10: $d^f_{exp,i-1}(e) \le d^0_{shift,i}(e)$ *and* $d^f_{shift,i}(e) \le d^0_{exp,i}(e)$.
Proof: Between the end of flow expansion at iteration i-1 and the start of flow shifting at iteration i, the auxiliary flow network changes through the elimination of $F(\delta C_i)$. The elimination of edges could eliminate existing paths and therefore $d(e)$ in the remaining residual capacity network can only increase. Since $Shift_i$ only adds edges $\sigma \rightarrow e$, the distances to τ in $Shift_i$ cannot decrease and therefore $d^f_{exp,i-1}(e) \le d^0_{shift,i}(e)$. For $Expand_i$ node distances to τ could further increase because flow reduction can only eliminate residual network edges present in $Shift_i$ for $f_{max,shift,i}$. Also, from Lemma 3 the addition of $F(\delta R_i)$ cannot reduce distances since it cannot add any edge from an event in $Shift_i$ to one in δR_i. Therefore, $d^f_{shift,i}(e) \le d^0_{exp,i}(e)$.□

Note that the argument in Lemma 10 does not hold if events are added in arbitrary order. In this case the addition of edges can reduce the distance function of some node e between a shifting and an expansion phase. In the worst case, this may reduce some distance back to a one unit distance for each application of maximum flow and therefore does not improve on the staged algorithm in (Muscettola, 2002).

Finally, consider reusing distance labeling across preflow-pushes for shifting and expansion. $\underline{d}^0_{shift,i}$, $\underline{d}^f_{shift,i}$, $\underline{d}^0_{exp,i}$ and $\underline{d}^f_{exp,i}$ are the distance labelings at the beginning and end of shifting and expansion. Assume also that the distance label of a node that has not yet entered $Expand_i$ or $Shift_i$ is zero.

Lemma 11: $\underline{d}^0_{shift,i}$ *can be made equal to* $\underline{d}^f_{exp,i-1}$ *for all nodes in* $Shift_i$. *Also,* $\underline{d}^0_{exp,i}$ *can be made equal to* $\underline{d}^f_{shift,i}$ *for all nodes in* $Expand_i$.
Proof: The distance label of a node remains valid when edges are deleted or new added edges always enter it from new nodes. Also, at node e the value of a distance function $d(e)$ must be an upper bound of that of the corresponding labeling $\underline{d}(e)$. From Lemma 10 we know that the distance function can only increase from $Expand_{i-1}$ to $Shift_i$ and from $Shift_i$ to $Expand_i$. Therefore $\underline{d}^f_{exp,i-1}$ and $\underline{d}^f_{shift,i}$ are valid choices respectively for $\underline{d}^0_{shift,i}$ and $\underline{d}^0_{exp,i}$.□

We can now prove the main complexity result.

Theorem 12: ***Incremental_Resource_Envelope*** *has time complexity* $O(Maxflow(n, m, U))$ *when one of the algorithms in Table 1 is used for flow contraction and flow expansion,*
Proof: Assume we applied one of the maximum flow algorithms in Table 1 to the resource increment flow network for the entire flexible plan (e.g., to compute the maximum envelope level over the entire time horizon (Muscettola, 2002)). This *full maximum flow* calculation is $O(Maxflow(n, m, U))$. We use Lemmas 9, 10 and 11 to prove that during envelope construction the cumulative cost of using the same algorithm for flow shifting and flow expansion is also $O(Maxflow(n, m, U))$.

1. *Labeling and Capacity scaling*: Lemma 9 shows that the worst case bound for the total flow moved during shifting and expansion is at worst twice the full maximum flow. Also, during shifting and expansion the cost of finding an augmenting path is at most m, the same of finding an augmenting path when computing the full maximum flow. Hence shifting and expansion cost at most $O(Maxflow(n, m, U))$.
2. *Successive shortest paths*: the cost of performing a single flow augmentation during full maximum flow calculation is an upper bound of the cost of a flow augmentation during shifting and expansion. The algorithm's complexity also depends on the monotonic increase of the distance function up to n after each elementary operation. Note that until the deletion of a δC_i or a P_{max}, a node's distance is bound by n, the same as during the computation of the full maximum flow. Monotonic increase is guaranteed by the algorithm

within each shifting and expansion phase and by Lemma 10 across phases. Hence, the cost is *O(Maxflow(n, m, U))*.

3. *Preflow-push methods:* the complexity is found through amortized analysis (Goldberg and Tarjan, 1988), relying on an appropriate potential function Φ and on the determination of its possible variations after the applying a local operation (e.g., a saturating or a non-saturating preflow push). One key observation is the monotonic increase of each node's distance label for each local operation. Both for the incremental and for the full flow this increase is bound by $2n-1$ and Lemma 11 guarantees monotonic distance label increase across phases. Note that, unlike for the computation of the full maximum flow, for shifting and expansion Φ increases also at the beginning of each shifting phase, when nodes are activated by the creation of initial flow excesses. However, a detailed amortized analysis (omitted for space limitations) shows that this increase does not affect the order of complexity of the shifting and expansion phases that remains *O(Maxflow(n, m, U))*.

The worst case complexity of the other phases of ***Incremental_Resource_Envelope*** besides shifting and expansion are dominated by *O(Maxflow(n, m, U)*. Flow reduction is cumulatively *O(nm)*. The total cost of **Extract_P_max** and of incrementally constructing and deleting the flow network is 2 *O(m)*. Finally the sorting of events during initialization is *O(n log n)*.□

Optimized Preflow-Push Implementation

For preflow-push implementations of the method, the previous complexity analysis indicates that we need to reuse the distance labeling function from the end of a maximum flow computation to the start of the next.

A further optimization is possible. Consider the maximum flow calculation on $Shift_i$. During initialization, an excess flow is loaded on each event e for each edge $\sigma \rightarrow e$ in $Shift_i$. We know that only a fraction of this excess flow may reach τ. The remainder will be pushed back out of $Shift_i$ during flow shifting and then pushed again through the flow network during flow reduction. In other words, this flow travels *twice* through the network before being eliminated. We can remove this duplication as follows. Assume that, instead of deleting the edges $\sigma \rightarrow e$ of $\underline{F}_{i-1}$ when constructing $Shift_i$, we delete the edges $\sigma \rightarrow e$ of $Shift_i$ after having performed the appropriate initial excess loading needed to perform flow shifting. In this case the flow that cannot be shifted will be pushed back to the source in $\underline{F}_{i-1}$, i.e., the source of $F(R_i)$, making the additional $O(nm)$ cost of flow reduction unnecessary. Another possible optimization consists of combining preflow-push through $Shift_i$ and $Expand_i$ by connecting δR_i *before* running the shift/reduce preflow-push. In this case the initializations of contraction and expansion are combined and a single preflow-push is run during phase *i*.

These optimizations do not affect asymptotic complexity but may have a significant effect in practice.

Conclusions

We presented a new class of algorithms that efficiently compute resource envelopes for flexible plans. The methodology is *O(Maxflow(n, m, U))* where n and m measure the size of the activity plan and U measures the maximum resource consumption or production of an event.

An empirical study over all algorithms of the class can shed light on the actual practical advantage of the incremental methodology versus the staged approach of (Muscettola, 2002) or the incremental method in (Kumar, 2003). While we expect that for large problem sizes the $O(n)$ cost reduction will be evident, practical improvements on smaller problems may require careful design of efficient and minimal data structures. Also, performance differences are likely to occur among implementations that use different maximum flow algorithms.

By shedding more light on the fine-grain structure of flexible plans, incremental envelope construction is also likely to improve the "benefit" part of the cost/benefit tradeoff in the use of envelopes for flexible scheduling.

Acknowledgements

Jeremy Frank, David Rijsman and Greg Dorais provided helpful comments on previous versions of this paper. This work was sponsored by the Automated Reasoning element of the NASA Intelligent Systems program.

References

R.K. Ahuja, T.L. Magnati, J.N. Orlin, 1993. *Network Flows*, Prentice Hall, NJ.

R.K. Ahuja, M. Kodialam, A.K. Mishra, J.B. Orlin, 1997. Computational Investigations of Maximum Flow Algorithms, *European Journal of OR*, Vol 97(3).

T.H. Cormen, C.E. Leiserson, R.L. Rivest, 1990. *Introduction to Algorithms*. Cambridge, MA.

A.V. Goldberg, R.E. Tarjan, 1988. A New Approach to the Maximum-Flow Problem. *JACM*, Vol. 35(4).

T.K.S. Kumar, 2003. Incremental Computation of Resource-Envelopes in Producer-Consumer Models, *Procs. of CP2003*, Kinsale, Ireland.

P. Laborie, 2001. Algorithms for Propagating Resource Constraints in AI Planning and Scheduling: Existing Approaches and New Results, *Procs. ECP 2001*, Spain.

P. Morris, N. Muscettola, T. Vidal, 2001. Dynamic Control of Plans with Temporal Uncertainty, in *Procs. of IJCAI 2001*, Seattle, WA.

N. Muscettola, 2002. Computing the Envelope for Stepwise-Constant Resource Allocations, *Procs. of CP2002*, Ithaca, NY.

N. Pollicella, S.F. Smith, A. Cesta, A. Oddi, 2003. Steps toward Computing Flexible Schedules, *Procs. of Online-2003 Workshop at CP 2003*, Kinsale, Ireland, http://www.cs.ucc.ie/~kb11/CP2003Online/onlineProceedings.pdf